ECOLE
POLYTECHNIQUE
DE BRUXELLES

MA2-IRIF

[INFO-H413] - Heuristic Optimisation

Implementation Exercice 1

https://github.com/Mudezer/LOP-VND

Author
Loïc Bermudez Arroyo

Professor

Thomas Stützle

Christian Camacho-Villalón

2023-2024

Abstract

This report focuses on the implementation exercice which is an attempt of solving the Linear Ordering Problem through different algorithms and comparing their efficiency and solution quality.

This work is divided in two parts, the first one focusing on simple iterative improvement algorithms using only one perturbative method and the second one focusing on variable neighborhood descent algorithms which make uses of a sequence of perturbative methods. The comparisons between the different algorithms were made through the Wilcoxon paired test in order to highlight whether two compared algorithms are different or not statistically.

The results have shown that for single perturbation algorithms the solution quality depends on the three factors being the perturbation method, the improvement procedure and the initialisation method but the greatest of them being the perturbation with the exchange neighborhood operation being the best.

For the sequential ones (e.g. VND) the results were not outstanding only being at the same level as the single perturbation ones or slightly better for the best of the VND's.

Contents

1 Introduction

1.1 Introduction

The Linear Ordering Problem has a wide range of application in several fields, such as scheduling, sports, social sciences, and economics making it an active combinatorial research subject. Due to its combinatorial nature, it has been shown to be NP-Hard, thereby there is no precise algorithm for solving LOP instances efficiently. As of today, researchers continue to work on it, thus collecting a significant amount of experiment and data making easy to compare algorithms.

1.2 Specifications

In this experiment on Linear Ordering Problems we used the XLOLIB instance class which is a database composed of 78 instances of two possible sizes: n=150 and n=250. We focused resolving the instances with simple perturbative local search algorithms, i.g. iterative improvement algorithms. In order to compare different algorithms, the Wilcoxon Paired Test has been used by using the relative deviation between the results and the best knowns of the instances.

1.3 Objectives

This work has been split in two parts:

1. Implement iterative improvement algorithms. The algorithms differentiate themselves from the initial solution generation, the improvevement procedure and the neihbourhood perturbation method. After generating the results, it is asked to give the statistics about the different algorithms such as the average deviation from the best knowns and the total computation time across all instances. And finally through a statistical test, show if there is a sta-

tisticall difference between the solutions generated by the different perturbative local search algorithms.

2. Implement a variable neighborhood descent algorithm considering two possible neighborhood relations, either : **transpose, exchange, insert** or **transpose, insert, exchange**. As with the iterative improvement part, it is asked to give the statistics and perform a statistical test in the same way.

1.4 Methods

For the instance solving a total of 14 algorithms were implemented, 12 iterative and 2 variable neihbourhood descent, being all combinations of:

- Two initial solution generation: *Random permutation* and *Chenery and Watanabe Heuristic*.
- Two improvement procedures: *best improvement* and *first improvement*.
- Three neighborhood perturbation: *exchange*, *insert* and *transpose*.

For the statistical tests, we opted fo the Wilcoxon signed-rank test (Wilcoxon paired test) which allows to show whether there is a statistical difference between the results of two different algorithms. A statistical test is based on a statement to be tested, in the Wilcoxon paired test this statement also said the null hypothesis is that the median of the differences between two sets is zero".

For the test a significance level α determining the maximum allowable probability of incorrectly rejecting the null hypothesis is chosen. Here we've chosen a α of 0.05 (typically the value are 0.05 or 0.01). This α will be compared to the p-value which the test returns, the p-value represents the probability that the null hypothesis is incorrectly rejected. If the p-value is smaller than the significance level, it means that the probability of falsely rejecting the null hypothesis is accepted.

In other words the smaller the p-value the more it shows that there is a difference between both sets. As the significance level plays a role of a threshold, if the p-value it this report is under 0.05 it then means that there is a statistical significant difference between the values we are observing.

2 Iterative Improvement

2.1 Statistics

In this section we have the statistics on all twelves iterative improvement algorithms. On table 2.1, the left table shows the results on the instances of size 150 and on the right table shows the results on the instances of size 250.

Algorithm	Size	Time	Deviation
iter_best_exch_cw	150	4.544384	3.6286836
iter_best_exch_rand	150	4.690247	3.6433558
iter_best_insert_cw	150	1.992183	5.0124972
iter_best_insert_rand	150	2.06022	5.2179703
iter_best_trans_cw	150	0.154221	22.530389
iter_best_trans_rand	150	0.132257	35.555230
iter_first_exch_cw	150	20.505894	2.7250084
iter_first_exch_rand	150	24.206485	2.9505165
iter_first_insert_cw	150	11.95989	4.7334026
iter_first_insert_rand	150	15.21029	5.4290908
iter_first_trans_cw	150	0.11861	22.618142
iter_first_trans_rand	150	0.103298	35.657041

Algorithm	Size	Time	Deviation
iter_best_exch_cw	250	48.51333	3.4733713
iter_best_exch_rand	250	51.9597	3.5704923
iter_best_insert_cw	250	21.310624	5.0487017
iter_best_insert_rand	250	23.341316	5.0254061
iter_best_trans_cw	250	0.591385	20.689539
iter_best_trans_rand	250	0.549525	33.196878
iter_first_exch_cw	250	296.59919	2.5485771
iter_first_exch_rand	250	344.57318	2.6877217
iter_first_insert_cw	250	163.33257	4.7446339
iter_first_insert_rand	250	215.62729	5.2765668
iter_first_trans_cw	250	0.45076	20.781501
iter_first_trans_rand	250	0.406784	33.298616

Table 2.1: Left table: Instances size of n=150, Right table: Instances size of n=250

Something notable already at this stage is that the deviations of algorithms using the transpose operations have a clear relative deviation from the best knowns, thus showing their clear lower quality in term of solution.

2.2 Statistical Tests

In this section, we will show the results of the different statistical tests that have been performed. we focused on comparing the different elements making up the algorithms, e.g. comparing initialization methods

2.2.1 Initial Solution

Hereafter we focused on comparing the initializations methods, the comparison is done based on their configurations. In short, we've compared algorithms only differing by their initializations.

CW	CW.Deviation	Random	Random.Deviation	p.value
iter_best_exch_cw	3.62868365465375	iter_best_exch_rand	3.64335585842101	0.928548201754893
iter_best_insert_cw	5.01249727445629	iter_best_insert_rand	5.21797037429794	0.0848496091239214
iter_best_trans_cw	22.5303893012667	iter_best_trans_rand	35.5552302318571	3.63797880709172e-12
iter_first_exch_cw	2.72500849330751	iter_first_exch_rand	2.95051657957667	0.0561791277541489
iter_first_insert_cw	4.73340263877678	iter_first_insert_rand	5.42909083703441	0.00117936985770939
iter_first_trans_cw	22.6181428895067	iter_first_trans_rand	35.6570413598824	3.63797880709172e-12

Table 2.2: Comparison of Initialisation on instances of size 150

CW	CW.Deviation	Random	Random.Deviation	p.value
iter_best_exchange_cw	3.47337136640244	iter_best_exchange_random	3.57049230707097	0.322156130601798
iter_best_insert_cw	5.04870173845386	iter_best_insert_random	5.02540611679944	0.643712874083576
iter_best_transpose_cw	20.6895392510665	iter_best_transpose_random	33.19687885281	3.63797880709172e-12
iter_first_exchange_cw	2.54857717496414	iter_first_exchange_random	2.68772179729456	0.0322441867356247
iter_first_insert_cw	4.74463397922102	iter_first_insert_random	5.27656689916324	9.09347581909971e-05
iter_first_transpose_cw	20.7815012756384	iter_first_transpose_random	33.298616377774	3.63797880709172e-12

Table 2.3: Comparison of Initialization on instances of size 250

From the table 2.2 we can see that appart for the insert neighbourhood in case of best improvement and for the echange neighbourhood there are significant differences when using one or other initialisation method.

From the table 2.3 we can see that only the insert neighbourhood and exchange neighbourhood in case of best improvement procedure hold little difference between both initialisation.

From the results when comparing the deviations and looking at the p-values we can see that the Chenery and Watanabe initialisation tends to gives a better solution quality even slightly and seem to be the best initialisation method.

2.2.2 Pivot Algorithm

In this section, we focused on the improvement procedure. In the same way that we compared the initialisations, we only compared algorithms differentiating by their improvement procedure only.

Best	Best.Deviation	First	First.Deviation	p.value
iter_best_exchange_random	3.64335585842101	iter_first_exchange_random	2.95051657957667	1.89866477739998e-06
iter_best_insert_random	5.21797037429794	iter_first_insert_random	5.42909083703441	0.13107475153447
iter_best_transpose_random	35.5552302318571	iter_first_transpose_random	35.6570413598824	0.0142851695636637
iter_best_exchange_cw	3.62868365465375	iter_first_exchange_cw	2.72500849330751	4.00177668780089e-10
iter_best_insert_cw	5.01249727445629	iter_first_insert_cw	4.73340263877678	0.020500839666056
iter_best_transpose_cw	22.5303893012667	iter_first_transpose_cw	22.6181428895067	0.0580596397303453

Table 2.4: Comparison of pivot procedure in instances of size 150

Best	Best.Deviation	First	First.Deviation	p.value
iter_best_exchange_random	3.57049230707097	iter_first_exchange_random	2.68772179729456	3.63797880709172e-12
iter_best_insert_random	5.02540611679944	iter_first_insert_random	5.27656689916324	0.123994226047216
iter_best_transpose_random	33.19687885281	iter_first_transpose_random	33.298616377774	1.56248643179424e-06
iter_best_exchange_cw	3.47337136640244	iter_first_exchange_cw	2.54857717496414	3.63797880709172e-12
iter_best_insert_cw	5.04870173845386	iter_first_insert_cw	4.74463397922102	0.00494850223185496
iter_best_transpose_cw	20.6895392510665	iter_first_transpose_cw	20.7815012756384	0.000302191027003574

Table 2.5: Comparison of pivot procedure in instances of size 250

For both instances size, the differences are significant only the insert with a random initialisation and the transpose with CW initialisation on instances of size 150 are quite similar.

When observing the deviations we can see that in general the first improvement procedure gives better results than the best one, appart for the algorithms using the insert with random initialisation where the best improvement procedure gives a better deviation and in both instances size. From what we see it seem that the first improvement procedure seem to be the best between both choices.

2.2.3 Neighbour Modification

And then in this last section about simple iterative improvement algorithms, the focus was on the neighbour perturbation methods. As done previously, the comparison were done in respect to the neighbourhood only.

Configuration	Exch.Dev	Inse.Dev	Trans.Dev	exch.ins.p.value	exch.trans.p.value	trans.ins.p.value
iter_best_random	3.6433558584	5.21797037429794	35.5552302318571	3.63797880709172e-12	3.63797880709172e-12	3.63797880709172e-12
iter_first_random	2.95051657957667	5.42909083703441	35.6570413598824	3.63797880709172e-12	3.63797880709172e-12	3.63797880709172e-12
iter_best_cw	3.62868365465375	5.01249727445629	22.5303893012667	2.00088834390045e-10	3.63797880709172e-12	3.63797880709172e-12
iter_first_cw	2.72500849330751	4.73340263877678	22.6181428895067	3.63797880709172e-12	3.63797880709172e-12	3.63797880709172e-12

Table 2.6: Comparison of neighbourhood modification on instances of size 150

Config	Exch.Dev	Inse.Dev	Trans.Dev	exch.ins.p.value	exch.trans.p.value	trans.ins.p.value
iter_best_random	3.57049230707097	5.02540611679944	33.19687885281	3.63797880709172e-11	3.63797880709172e-12	3.63797880709172e-12
iter_first_random	2.68772179729456	5.27656689916324	33.298616377774	3.63797880709172e-12	3.63797880709172e-12	3.63797880709172e-12
iter_best_cw	3.47337136640244	5.04870173845386	20.6895392510665	7.275957614183 44e-12	3.63797880709172e-12	3.63797880709172e-12
iter_first_cw	2.54857717496414	4.74463397922102	20.7815012756384	3.63797880709172e-12	3.63797880709172e-12	3.63797880709172e-12

Table 2.7: Comparison of neighbourhood modification on instances of size 250

The Wilcoxon test shows that there a big and significant differences between the three neighbourhood modification methods. When looking directly to the deviation values it is clear that the exchange method has a clear advantage whatever the rest of the configuration of the algorithm. Something to not is that the insert operation do not bring that bad of results, when compared with the transpose operation which give at minimum a deviation of 20 which is no small value.

3 Variable Neighbourhood Descent

3.1 Statistics

The following table resume the statistics of the variable neighbourhood descent experiment. The idea of the VND was to alternate in a certain way between the neighbourhood modification sequence in order to avoid getting stuck in a local maxima and trying to look to have a bette result.

Algorithm	Size	Time	Deviation
vnd_first_cw_TEI	150	9.979591	2.65762453832907
vnd_first_cw_TIE	150	8.100981	3.82220234217974

Algorithm	Size	Time	Deviation
vnd_first_cw_TEI	250	125.65971	2.49148964131548
vnd_first_cw_TIE	250	101.59698	3.69107733591532

Table 3.1: Left table: Statistic of instances of size 150, Right table: Statistics of instances of size 250

3.2 Neighbour Sequence

The sole way to compare both algorithms was looking to the neighbourhood modification sequences, two were used:

- Transpose, insert, exchange - TIE
- Transpose, exchange, insert - TEI

No other parameter were changed.

TEI	TEI.Deviation	TIE	TIE.Deviation	p.value
vnd_first_cw_TEI	2.65762453832907	vnd_first_cw_TIE	3.82220234217974	3.63797880709172e-12

Table 3.2: comparison of neighbour perturbation sequence in instances of size 150

TEI	TEI.Deviation	TIE	TIE.Deviation	p.value
vnd_first_cw_TEI	2.49148964131548	vnd_first_cw_TIE	3.69107733591532	3.63797880709172e-12

Table 3.3: comparison of neighbour perturbation sequence in instances of size 250

The experiment shows clearly a big difference between both sequences, where the TEI gives better results.

3.3 Comparison with Iterative Improvement

The VND algorithms following the TEI sequence has a clear better results than the other algorithms, even better than the simple iterative algorithms using the exchange neighborhood. The TIE sequence present slightly worst or same results than the exchange neighborhood in some configurations such as the best_random or best_cw.

This highlights the fact that the exchange neighborhood method is generaly better at finding good solutions than the transpose or insert.

4 Conclusion

4.1 Iterative Improvement

By comparing the different element composing the simple iterative algorithms the results show clearly that the Chenery and Watanabe initialisation gives the best results overall compared to the random one. The exchange neighbour operation also gives the best results when compared to the insert and transpose. And the first improvement gives the best results as well when compared to the best improvement procedure. This is confirmed directly by looking to the table 2.1 when looking to the first_exch_cw algorithms.

4.2 Variable Neighbourhood Descent

Unlike what expected, the VND algorithms did not have skyrocket differences with the best simple iterative algorithm. This could be explained by the fact that each sequence starts with the transpose neighborhood which gives the worst results in term of deviations. Nevertheless, the TEI sequence brings out better results than the first_exch_cw one, this could come from the fact that having a sequence of multiple neighborhood operations allows to effectively get out of local optimum and to seek better results.

Now in term of quality between the sequences, the results clearly show that the TEI gives better results than the TIE which might come from the fact that in the TIE the exchange neighbourhood comes lasts in the sequence thus allowing much less the algorithm to profit of the exchange neighborhood efficiency and thus leading to getting stuck to a lower quality of local optimum.